COMP 310
Winter 2024
Assignment 1

Posted: Monday, January 22

Due: Wednesday, February 7th, 11:59 p.m.

Question 1:	100 points
	100 points total

Please read the entire PDF before starting. It is very important that you follow the directions as closely as possible. The directions, while perhaps tedious, are designed to make it as easy as possible for the TAs and automated tests to run your code. **This assignment will count for 10% of your final grade.**

To get full marks, you must follow all directions below:

- You must write your code in the C programming language. We use C in this class because most of the practical contemporary operating systems kernels are written in C/C++ (e.g., Mac OS, Linux, Windows, Linux and many others).
- Make sure that all file names and function names are **spelled exactly** as described in this document. Otherwise, a 50% penalty per question will be applied.
- Make sure that your code **compiles**. Code that does not compile will receive a very low mark, and possibly a zero.
- Write your name and student ID in a comment at the top of all code files you hand in.
- Your code must follow our style guidelines, which are listed on the next page. **Up to 30% can be removed for code that does not comply to our style guidelines.**

Hints & tips

- **Start early.** Programming projects always take more time than you estimate!
- Do not wait until the last minute to submit your code. **Submit early and often**—a good rule of thumb is to submit every time you finish writing and testing a function.
- Write your code **incrementally**. Don't try to write everything at once. That never works well. Start off with something small and make sure that it works, then add to it gradually, making sure that it works every step of the way.
- Seek help when you get stuck! Check our discussion board first to see if your question has already been asked and answered. Ask your question on the discussion board if it hasn't been asked already. Talk to a TA during office hours if you are having difficulties with programming. Go to the instructor's office hours if you need extra help with understanding a part of the course material.
  - Note that small amounts of code can be posted on the discussion board as *private* posts, which only the instructors and T.A.'s can see. But if more than a few lines of code are in question, then it is better to seek help in person. Often the problem requires a different *approach* such as proper use of the debugger, and the discussion board is a poor medium for this kind of help. (You could also post a single line of code and error message (error traceback) as a public post.)

- If you come to see us in office hours, please do not ask ``Here is my program. What’s wrong with it?'' We expect you to at least make an effort to start to debug your own code, a skill which you are meant to learn as part of this course. Reading through someone else’s code is a difficult process—we just don’t have the time to read through and understand even a fraction of everyone’s code in detail.
  - However, if you show us the work that you've done to narrow down the problem to a specific section of the code, why you think it doesn't work, and what you've tried in order to fix it, it will be much easier to provide you with the specific help you require and we will be happy to do so.

Policy on Academic Integrity

See section 5 of our syllabus for our course policies on academic integrity. In short, you must write the assignment on your own. Do not copy someone else's work, nor share your work with someone else. Do not copy from any online sources.

Late policy

Late assignments will be accepted up to two days late and will be penalized by 10 points out of 100 per day. Note that if you submit one minute late, it is equivalent to submitting 23 hours and 59 minutes late, etc. So, make sure you are nowhere near that threshold when you submit.

Revisions

Search for the keyword *updated* to find any places where the PDF has been updated.

Style guidelines

As students at the 300 and 400 level, it is expected that you code at the level of a beginner software engineer. Please follow the brief coding style below in all your OS assignments.

- Indentation: All blocks of code must be indented by a minimum of 4 spaces, or with a 1-tab. (Pretend you are writing Python code.)
- Horizontal spacing: Parts of the code that are logically different should be separated by a blank line.
- Comments: Provide useful explanations to the reader of the code. We do not expect you to comment all your code, but we do expect explanations for more complex parts. In other words, do not write comments that just restate the code; here are some situations where a comment may be appropriate:
  - describing the purpose of a function
  - explaining the high-level steps of a function
  - explaining magic numbers
  - describing side-effects of a function
- Names: Name your variables, constants, functions, objects, methods, data structures and files appropriately. The purpose of each should be obvious from the name and can save you from writing a comment.

Compilation environment

Because of the many different versions of gcc and C library implementations, we will ask you to use a standardized testing environment when compiling and testing your code. We will be using the same environment when grading your code, so it's important that we all use the same one. If your code does not compile in this environment, you will receive a very low mark, and possibly a zero.

We will use **Docker** for our testing environment. Here are the steps you should follow:

1. Install Docker by following the steps at `https://docs.docker.com/get-docker`. Docker is available for Mac, Linux and Windows.

   Note: You do not need to sign up for a Docker account. You can skip that step.

2. Open up a command line and enter the following command:

   ```
   docker pull gcc:13.2
   ```

3. `cd` to the directory containing your assignment code files (*.c, *.h and Makefile). Then enter the following command to run a Docker shell. This command is long, so you may want to alias it:

   ```
   docker run --rm -it --mount type=bind,source=.,target=/code --entrypoint /bin/bash -w /code gcc:13.2
   ```

4. Once you are in the Docker shell, you should find yourself in the `/code` directory by default. This directory will contain all the files in the folder you were in before launching the shell. You can now run `make` to compile your code.

   Note: Any changes made to your files here (including the creation of your executable file) **will** be reflected in your original folder (on your hard drive). So, be careful not to delete any files from within Docker.

Examples

For each question, we provide **examples** of how your code should behave. All examples are given as if you were to run the commands while inside the shell.

During grading, we will run these examples and verify that your code outputs the same as given in the examples. Part of your assignment mark will be determined by the results of these tests. We will also run additional, private tests on your code. Therefore, it is your responsibility to make sure your code/functions work for any inputs, not just the ones shown in the examples.

Submitting your code and running the public tests

To run your code on our public tests, as well as to submit your code for grading, you will need to follow certain instructions. These will be conveyed to you in the coming days.

Question 1: Your very own shell (100 points)

Welcome to the first assignment in Operating Systems!

In this assignment, we will build our own OS shell.

Starter files

To get you started, we provide a starting template for your shell, which you will enhance.

Herein we describe how to compile and run the starter code, what commands the starter code already contains, and then what commands you will need to implement:

Compiling

- You will compile your shell using the command `make myshell`
- After making changes, make sure to run `make clean; make myshell`
- You will compile in the Docker environment as detailed earlier in this PDF.

Running

There are two ways to run your shell:

- **Interactive mode**: Simply type `./myshell` at the prompt to run the executable and be dropped into the shell.
- **Batch mode**: Type `./myshell < commands.txt` to give your shell a text file (in the current directory) containing a list of shell commands (one line per command). Your shell will open the file and run each command, one at a time, starting from the first line of the file. You will not be dropped into an interactive shell. If an error occurs in processing of a command, an error message will be displayed and the shell will move to the next command.

Existing commands

Your shell already supports the following (case-sensitive) commands:

- `help`: displays a list of the commands
- `quit`: exits the shell
- `set VAR STRING`: assigns a value to a variable in shell memory (if the variable already exists, its value will be overwritten). Values can be a single alphanumeric word without special characters nor spaces.
- `print VAR`: displays the value assigned to the specified variable. If the variable does not exist, an error message will be displayed.
- `run SCRIPT.TXT`: runs the commands in the given file in batch mode (see above). Assumes that the file exists.

If the user enters an invalid command, an error message will be shown.

Your tasks

You must add the following functionality to your shell:

1. **Enhance the `set` command.** The `set` command as described above assumes that the STRING value is a single alphanumeric token without special characters nor spaces. You must extend the `set` command to support a value that contains between 1 and 5 alphanumeric tokens (inclusive).

   Notes:

   - If there are less or more tokens than in that range, your shell should not set the new value and instead print out the error message `"Bad command: set"`
   - You can assume that tokens will be separated by a single space.
   - You can assume that a single token will be no larger than 100 characters.

   Example (in interactive mode):

   ```
   $ set x 10
   $ print x
   10
   $ set x 20 bob alice toto xyz
   $ print x
   20 bob alice toto xyz
   $ set x 12345 20 bob alice toto xyz
   Bad command: set
   $ print x
   20 bob alice toto xyz
   ```

2. **Add the `echo` command.** The echo command is used for displaying strings which are passed as arguments on the command line. This simple version of `echo` only takes one token string as input. The token can be one of two types:

   - an alphanumeric string. In this case, `echo` simply displays the string on a new line.
   - an alphanumeric string preceded by a $. In this case, `echo` checks the shell memory for a variable that has the name of the alphanumeric string following the $ symbol. If it is found in memory, the associated value should be displayed to the user (similar to the `print` command). If it is not found, an empty line should be displayed instead.

   Notes:

   - You can assume that the token string will be no larger than 100 characters.

   Examples (in interactive mode):

   ```
   $ echo mary
   mary
   $ echo $mary

   $ set mary 123
   $ echo $mary
   123
   ```

3. **Add the `my_ls` command.** It should list all the files present in the current directory. Note that you can invoke the system's `ls` command to accomplish this quite easily.

4. **Add the `my_mkdir dirname` command.** It should create a new directory called `dirname` in the current directory. You can assume that `dirname` will be an alphanumeric string of no longer than 100 characters.

5. **Add the `my_touch filename` command.** It should create a new empty file inside the current directory with the given filename (which you can assume will be an alphanumeric string of no greater than 100 characters).

6. **Add the `my_cd dirname` command.** It should change the current directory to the specified directory. If the directory does not exist, then instead an error message `"Bad command: my_cd"` should print out. You can assume that `dirname` will be an alphanumeric string of no longer than 100 characters.

7. **Add the `my_cat filename` command.** It should open the file at the given filename and print its contents to the screen. You can assume the filename will be an alphanumeric string of no greater than 100 characters. If the file does not exist, then instead an error message `"Bad command: my_cat"` should print out.

8. *Optional: The following part is not worth any marks and will not be graded. However, if you pass the tests for this part, you will be awarded 1,000 `COMP310COIN` when you submit your code and pass the corresponding public test.* **Add support for `if` conditionals.** An `if` conditional will have the following syntax:

   ```
   if identifier op identifier then command1 else command2 fi
   ```

   The `identifier`'s can be either single alphanumeric tokens (of no more than 100 characters) or a variable beginning with a `$` (in which case you should use the value of the variable stored in memory, like you did with the `echo` command). The `op` can be either `==` (equals) or `!=` (not equals). If the condition (`identifier op identifier`) is true, then `command1` should be executed; otherwise, `command2` should be executed.

   Commands can be any series of alphanumeric tokens until the `else` or newline.

9. *Optional: The following part is not worth any marks and will not be graded. However, if you pass the tests for this part, you will be awarded 1,000 `COMP310COIN` during grading.* **Add another command to the shell.** Shells today have many useful commands. Choose a command, or program you think would be useful to be a shell command, and implement it into your shell. Do not choose a one-liner, but instead something that requires a bit of time to implement. Our TAs will judge its appropriateness to be given coins during grading. (You can ask on the discussion board if you are unsure.)

Finally, add a **`README.md`** file into the same folder as your code files. In this file, state your name(s) and McGill ID(s), any comments you would like the TA to see, and whether you have implemented the second optional part (adding an extra command), and if so, the name of the command, so that the TA will know to check for this.